Implement continue and abort command

Organization: **Python Software Foundation**

Sub-Org: **Mercurial**

Sub-Org: **Mercurial**

I want to work with **Mercurial**.

For almost all software projects, the source code is like the crown jewels - a precious asset whose value must be protected. Version Control Systems protects source code from both catastrophe and the casual degradation of human error and unintended consequences. Their role is to manage the development and maintenance of software projects. It allows you to work without worry and experiment with your code while keeping the stable version untouched.
Mercurial is a free and distributed version control system. Its interface is very intuitive, and it scales to projects of any size.

About Me

Basic Information

Name	Taapas Agrawal
University	Indian Institute of Technology Kharagpur
Email	taapas2897@gmail.com
IRC	taapas1128 at freenode.net
Bitbucket	taapasX28
Blog	https://taapasx28.github.io/
Timezone	IST (UTC +5:30)

Platform Details

OS	Ubuntu 18.04

Editor Visual Studio Code

Personal Background

I am a second-year undergraduate student at IIT Kharagpur, India. I'm pursuing a major in Applied Geology and a minor in Mathematics and Computing. I've been coding in C and C++ for over two years and in Python for a year and am proficient in all three of them now.

I like Python because it lets me convert my ideas into code. It is more flexible and robust. I am also a robotics enthusiast and work for the Kharagpur Robosoccer Students' Group (KRSSG) of my college. It is a group that participates in an annual international competition called RoboCup. I have worked on Reinforcement Learning based framework for humanoid bots using Python and its libraries. Python helps me prototype algorithmic implementations with much less effort in comparison to other programming languages where developing such models will be difficult.

I am quite familiar with version control systems and have previously used both Mercurial and Git for maintaining personal projects as well as organization based projects. I am quite fluent with Mercurial.

Contributions to Mercurial

The top is the oldest

[**MERGED**] push: add "remote" to 'repository changed while pushing' messages (issue5971)

This patch makes the message while pushing to remote repository clearer so the user is not confused whether the repository is local or remote.

[**MERGED**] amend: added config option to update time to current in hg amend(issue5828)

This adds a new config option to mercurial called `rewrite.update-timestamp` to `hg amend` which optionally updates date to current when *True*. Further, when `--date` flag is specified along with the new config option then `--date` is given priority over the config option.

[**MERGED**] histedit: add rewrite.update-timestamp support to fold and mess

This extends the support of `rewrite.update-timestamp` to histedit fold and mess options. This also adds `tests/mockmakedate.py` for supplying testable values in case current time is invoked in the tests.

[**MERGED**] tests: replace mockmakedate function in test-amend.t

This is a follow-up patch to the previous one which integrates `mockmakedate.py` with tests regarding `hg amend`.

[**MERGED**] revert: add prompt before undeleting a file in -i (issue6008)

This patch adds the feature of a prompt that asks whether or not a removed file is to be undeleted in `hg revert` interactive mode.

[**MERGED**] uncommit: added interactive mode(issue6062)

This is the first attempt to add interactive mode for `hg uncommit.`

[**WIP**] D6005: uncommit: added interactive mode -i(issue6062).

This patch is to add the interactive mode to hg uncommit. This will be much cleaner and robust than the previous implementation. The basic logic is to first uncommit everything then performs an action similar to an interactive amend.

[**WIP**] D6038: push: added clear warning message when pushing closed branches(issue6080).

This patch makes the message while pushing new branches clearer. Earlier even if a new branch being pushed was a closed branch then to the message was same and contained no information about the closed branches. This patch determines the number of closed branches among the new branches and adds it to the warning message.

The Project

Abstract

Title: Implement continue and abort command

Description of the problem

In mercurial we have commands like graft, histedit, rebase, shelve etc. which when used might end up in a conflicted state which then requires the user to intervene and give commands which abort or continue the operation that is currently being executed.

There are pre-developed `--abort` and `--continue` flags which help the user to do so. However, the downside of this is that the user needs to remember the last command that he used. This project is all about implementing generic `hg continue` and `hg abort` commands which will automatically scan for the command that is being currently in conflict. Furthermore, this will provide with the functionality for extensions to plug in their logic to abort and continue the operation. With this feature added, out of core extensions will be able to use `hg continue` and `hg abort` commands.

Unifying of State Determining APIs

Mercurial already has APIs which helps in determining the state of multistep commands like graft, histedit, rebase, shelve etc. However, there are two such APIs present. As stated by **Pulkit Goyal** and **Pierre-Yves David** on IRC. One of them is used by `hg status --verbose` another one is used in `hg resolve` to abort a process in case of a conflict. These APIs need to unified into a single one and then extended to have a possibility to register commands that come under `abort` and `continue` functions.

Implementing the logic for hg update --abort

Mercurial has `hg update` command which updates the repository's working directory to the specified changeset i.e. it is a feature which helps you change though different versions of your code easily with just a single command. However, while updating to a specified changeset with a *dirty* working directory, conflicts can occur. Once the conflicts have occurred the user is hardly left with a choice. He must resolve the conflicts to get to work any further and he has no option to go back and *abort* the update. This has been one of the most requested features for mercurial which requires to create `hg update --abort` which will help the user to revert to the situation just before `hg update` command was given. After implementation, this too will be included in `hg abort` like the other commands stated above.

Overview

What does the pre-existing --continue and --abort flag do?

The names of these flags are quite self-explanatory; `--abort` and `--continue` flag which helps the user to abort a command that is being executed or continue the operation the way it was going on. They are generally used when the operation enters in a state that is questionable and requires the user needs to manually check it and depending on that the user may either wish to continue the operation or abort it. An example of a situation is when while importing code from a patch a user might enter in a condition involving a merge conflict. In such a condition there are two ways in which

he can proceed either the user should abort the operation and go back to the initial state or the user may resolve the merge conflict and then use the continue flag to continue the operation of importing the code.

How will be the utility of hg abort and hg continue?

There are problems that `hg abort` and `hg continue` will solve.

- This will add the feature that the user will not require to remember the previous command and attach the flag to it.
- This will add functionality for extensions to plug in their logic for aborting and continuing command. For example, commands introduced by out of core extensions will be able to use hg continue and hg abort functionality.
- The two APIs which determine which operation is active currently and performs their state check will be unified during the course of this project which will make the code even more organized and abstract.

Why --abort flag is needed for hg update when --check flag is present?

To deal with similar conditions where hg update may result in a merge conflict two different flags `--check` is present. This it is used with `hg update` the working directory is checked for uncommitted changes; if none are found, the working directory is updated to the specified changeset and if uncommitted changes are found then the update is aborted and the uncommitted changes are preserved.

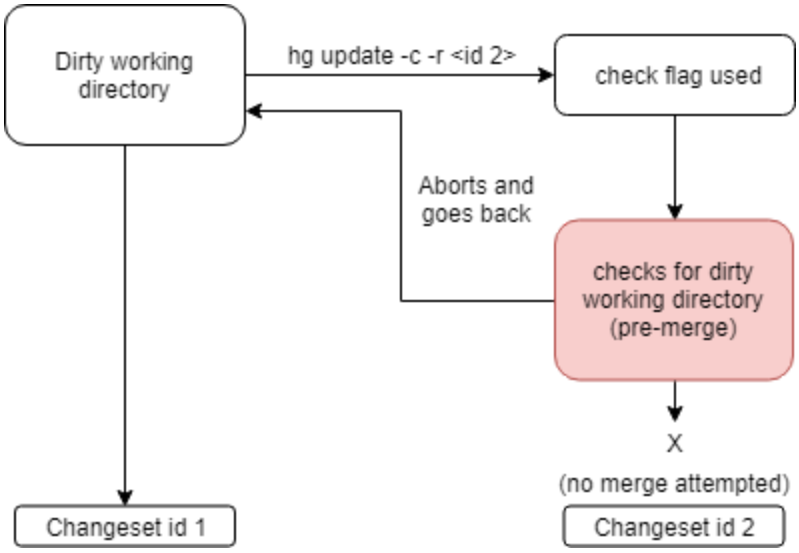


Figure 1

As stated in *Figure 1* using `--check` flag with update reverts to the original changeset without attempting a merge between the uncommitted changes with a new parent revision.

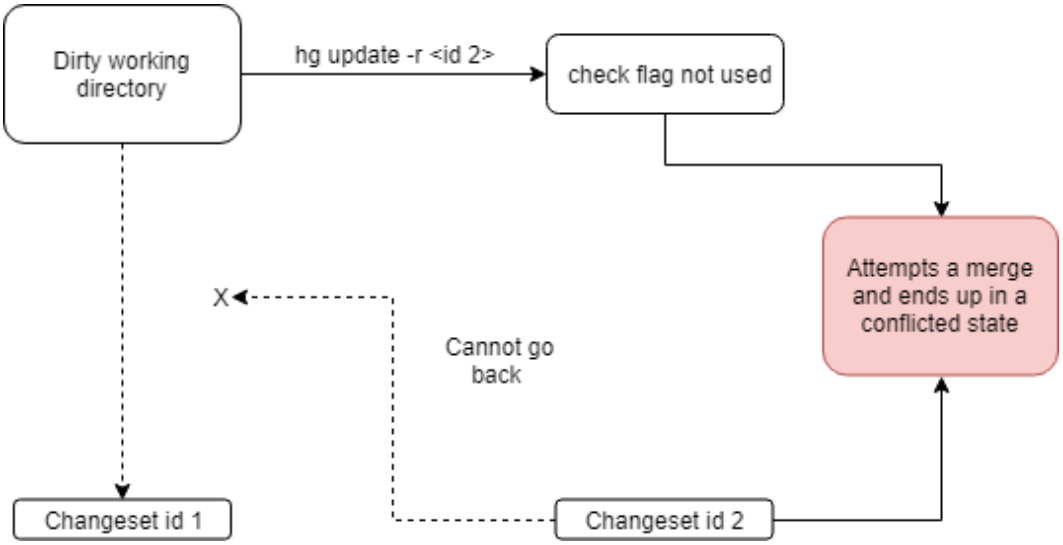


Figure 2

However, if the user forgets to use the `--check` flag then a merge is attempted and it leads to a conflicted state which cannot be reverted(*Figure 2*). `--abort` flag is to deal with such situations where a merge has already occurred and the user is in a conflicted state. This flag will help revert to the original state before the update was attempted.

Potential Problems

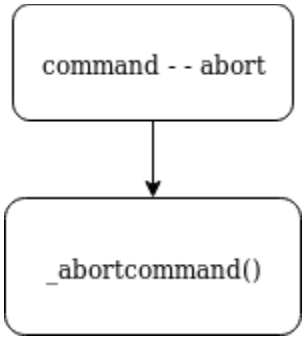
1. As stated above the two present APIs need to unified for the purpose of implementation of `hg abort` and `hg continue.` However, this will require understanding code that has been present in the core for long enough. Also after the unification of the APIs, it must be made sure that all the functions and commands which once called these APIs are made bug free.
2. Although in most of the commands which come under the project have well-defined logic to abort and continue a command but the call of these functions need to be re-diverted from the newly added `hg abort` and `hg continue` commands.

Implementation

The first step will be the unification of APIs. As far as I have analyzed the two APIs are currently located in `cmdutil.py`. The code for the first API as used by `hg status --verbose` lies in `cmdutil._getrepostate()` whereas that used by `hg resolve` lies

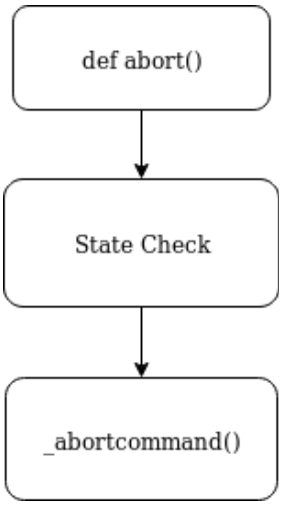
in `cmdutil.checkunfinished()`. Both of these use `repo.vfs.exists()` at a very basic level to check which command is active right now and the state of the command. These two APIs will require a deeper study and then using that plan a new unified API will be written.

Currently when a `--abort` flag is used along with the command name as `hg <command name> --abort` then the `_abortcommand()` is called which contains the logic regarding the abortion of the particular command. However, the command name is to be remembered in such a case.



The implementation I am thinking of does as follows. Whenever `hg abort` is called `def abort()` will be called which first will perform a state check and find out which command is currently active or in a conflicted state. Once that is determined then the respective `_abortcommand()` function is called and the process is aborted. APIs to check the state of a particular command is currently present but these APIs need to unified for the purpose of simplification. Furthermore, the `def abort()` command will be present in the core while the logic to abortion of a command i.e. `_abortcommand()` function with be present with the extension.

Also, direct calls to `_abortcommand()` function using `--abort` flags will be redirected to `def abort()` with appropriate arguments.



A very naive implementation of hg abort using the existing state determining API to combine abort for `hg graft` and `hg histedit` can be as follows:

```
@command('abort', [], helpcategory=command.CATEGORY_CHANGE_MANAGEMENT,
        helpbasic=True)
def abort(ui, repo, **opts):
    """this function is a prototype for hg abort command.

    It first checks the state and checks which command is active
    then calls then returns the respective logic for aborting the
    command.
    """
    graftstate = statemod.cmdstate(repo, 'graftstate')
    if graftstate.exists():
        return _abortgraft(ui, repo, graftstate)
    histeditstate = statemod.cmdstate(repo, 'histeditstate')
    if histeditstate.exists():
        return histedit._aborthistedit(ui,repo,histeditstate,nobackup=True)
```

`hg continue` will also use a similar approach to solving the problem once the APIs have been unified.

For hg update --abort

As stated by Gábor Stefanik (2016-11-02 14:05:55 UTC) in issue 4404 on Bugzilla, the implementation of `hg update --abort` would require the following workflow to be

combined under a single command: `hg resolve -u, hg resolve --all --tool internal:local, hg up <original revision>, hg resolve -u, hg resolve -all --tool internal:local`. This involves first marking the conflicted files as unresolved then selecting all unresolved files and then using the local p1() version of them as the merged version. After that, it needs to updated back to original version i.e the changeset before `hg update` was used. Then the earlier process is repeated from marking the files unresolved onwards.

Documenting the new behaviour of all the changes

Since this project will add two new commands, i.e. `hg abort` and `hg continue` there will be changes that haven't been documented yet. I will keep documenting all the small changes that I introduce as I progress through the summer. I will be writing a wiki at the end to highlight all the additions that were made throughout the summers. Also, I will update the manual and help for these two new commands added.

Writing Tests

As I'll be making a lot of changes in the core ranging from the unification of APIs to the addition of two new commands, there are good chances that I'll encounter a lot of bugs. I plan to resolve them and write necessary tests which will demonstrate their working. I am also planning to maintain a blog regarding my GSoC experience and the description of the code I am adding. I will update this blog weekly.

Timeline (Tentative)

Community Bonding period (May 6 - May 27)

I will utilize this period to develop a deeper understanding of the two APIs used to check the state of a command. I will also try to write a plan for the basic framework which will be generated after the unification. I will try to make that as efficient as possible.

Week 1 - Week 2 (May 27 - Jun 10)

I will start the unification of the APIs based on the framework that I concluded in the previous period. Also, this will also require cleaning up the code and changing the part where the previous API functions were called.

Week 3 - Week 4 (Jun 11 - Jun 24)

In this period I will start working on `hg abort` . I will extend its support to `hg graft.` Also in case time is left I will start adding support for the in-core extensions starting from `hg histedit` followed by `rebase` and then `shelve`.

Phase 1 Evaluation

Week 5 - Week 6 (Jun 25 - Jul 8)

I will complete the integration of in core extensions if left and then try to extend the support to `evolve` extension . I will also update the blog and the manual for `hg abort.`

Week 7 - Week 8 (Jul 8 - Jul 22)

In this period I will start working on `hg continue` . I will extend its support to `hg graft.` Also in case time is left I will start adding support for the in-core extensions starting from `hg histedit` followed by `rebase` and then `shelve`.

Phase 2 Evaluation

Week 9 - Week 10 (July 23 - Aug 5)

I will complete the integration of in core extensions if left and then try to extend the support to `evolve` extension. I will also update the blog and the manual for `hg continue.` If time is left I will start working on `hg update --abort.`

Week 11 - Week 12 (Aug 6 - Aug 19)

During this period I will implement the code for `hg update --abort.` I will extend its support to `hg abort.` I will update the documentation and blog.

Week 13 (Aug 20 - Aug 26)

Hopefully, I would have sorted everything by now. I will keep this week as a buffer and wrap up any backlogs that are left regarding any bug in the code or documentation. But if time is left, I will work on `hg stop`.

Post GSoC

I have improved my skills a lot as compared to the first time I started to contribute to mercurial. I learned how commercial coding is done and how precise, abstract and clean a piece of code can be. People like Pulkit, Martin, Yuya, Augie helped me realise this and helped me a lot throughout my contributions pointing even the silliest of my errors. I would like to continue my contributions to mercurial. Even after this project is over I would try to improve the code for `hg abort` and `hg continue` even further. I would also love to take up an idea from **WeShouldDoThat** page and keep on solving issues from Bugzilla. I plan to be even more active on the IRC channel and keep my learning curve high.

How do the mentors keep track of my work?

I will be posting my patches regularly via the Phabricator for review of the mentors. For the purpose of contributing to `evolve` I will be using BitBucket and I will be pushing all the codes there. In case it is necessary I will also use the mailing list to send the patches. Also, I will be very active on the IRC channel (#mercurial), where I will be discussing all the doubts and developments related to my project.

Other Commitments

I have no other commitments this summer. For GSoC 2019, I am only applying for Mercurial. So if selected, I will easily be able to dedicate about 40 hours a week to Mercurial. My college restarts in mid-July, but I will still be able to contribute full time. Furthermore even after my college reopens, I have no exams or tests during the GSoC period to hinder with my performance.